Scalable parallel out-of-core terrain rendering

Goswami, P ; Makhinya, M ; Bösch, J ; Pajarola, R

Abstract: In this paper, we introduce a novel out-of-core parallel and scalable technique for rendering massive terrain datasets. The parallel rendering task decomposition is implemented on top of an existing terrain renderer using an open source framework for cluster-parallel rendering. Our approach achieves parallel rendering by division of the rendering task either in sort-last (database) or sort-first (screen domain) manner and presents an optimal method for implicit load balancing in the former mode. The efficiency of our approach is validated using massive elevation models.

Eurographics Symposium on Parallel Graphics and Visualization (2010)
J. Ahrens, K. Debattista, and R. Pajarola (Editors)

Scalable Parallel Out-of-core Terrain Rendering

Prashant Goswami[†] Maxim Makhinya[‡] Jonas Bösch[§] Renato Pajarola[¶]

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

Abstract

In this paper, we introduce a novel out-of-core parallel and scalable technique for rendering massive terrain datasets. The parallel rendering task decomposition is implemented on top of an existing terrain renderer using an open source framework for cluster-parallel rendering. Our approach achieves parallel rendering by division of the rendering task either in sort-last (database) or sort-first (screen domain) manner and presents an optimal method for implicit load balancing in the former mode. The efficiency of our approach is validated using massive elevation models.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed Graphics; I.3.m [Computer Graphics]: Miscellaneous—Parallel Rendering

1. Introduction

Interactive visualization of very large terrain datasets remains a challenging task as precision of data acquisition is increasing with the advancement of hardware. Moreover, in many visualization applications terrain rendering itself is only a secondary task and is often relegated to the background. The massive sizes of terrain data and the demand of high quality data display make it challenging for the application to run at interactive frame rates even with the best hardware on a single machine. Therefore, there has been an inclination towards the use of parallel rendering approaches, which allows one to use resources from multiple systems, improving the rendering capacity manifold.

Continuing improvements in CPU and GPU performance as well as increasing availability of multi-core processors and cluster-based parallel systems demand for flexible and scalable parallel rendering solutions that can exploit multi-pipe graphics hardware. However, such solutions are not always easy to design and often not inherent in the basic structure or choice of algorithms. Our goal in this work is to come up with an efficient out-of-core parallel and scalable terrain rendering approach that allows interactive visualization of huge digital elevation models (DEMs) while still maintaining the desirable features and properties of state-of-the-art terrain rendering algorithms.

Many algorithms have been proposed for interactive rendering of large DEMs, limiting the 3D graphics load effectively to the available computing and graphics resources as well as display needs. To improve rendering performance, an appropriate level-of-detail (LOD) of the graphics data to be displayed is selected for each frame. The LOD is adaptive with respect to surface features and viewing parameters and is selected to achieve a target rendering quality and frame rate. With current generation CPU-GPU configurations, emphasis is more on optimizing LOD over batched primitives than on processing individual geometric primitives such as vertices or triangles. This way, a coarse LOD selection is quickly accomplished by the CPU and then the corresponding geometry is handed over to the GPU for batched rendering. Hence, the CPU does not consume excessive time to perform fine-grained LOD optimizations that risk starving the fast-paced graphics hardware pipeline.

Many of the state-of-art terrain rendering algorithms support high-performance rendering and compact storage of

[†] e-mail: goswami@ifi.uzh.ch
[‡] e-mail: makhinya@ifi.uzh.ch
[§] e-mail: boesch@ifi.uzh.ch
[¶] e-mail: pajarola@acm.org

DEM data. However, not only do these approaches lack simplicity of implementation and usage, but they also do not keep DEM height data in a format independent from the multiresolution triangulation. These approaches do not allow the DEM data to be used by other non-rendering applications, thus requiring data duplication. Even more, many methods cannot guarantee continuous out-of-core LOD fetching and display to maintain a consistent frame rate. Most importantly, none of these algorithms can be parallelized easily and efficiently across multiple computer systems and distributed graphics resources.

The presented parallel terrain renderer is built upon RASTeR [BGP09], which introduced the concept of a paired multiresolution tree structure where the multiresolution triangulation is independent of the DEM data. In order to achieve this, the DEM data is kept in square tiles, which are the basic units of a quadtree, and the multiresolution triangulation is organized as triangle patches which are the basic units of a meta triangle-bintree. Triangulation itself is never explicitly stored and is based on a lean LOD data structure. Continuous display is achieved through out-of-core DEM and texture data fetching by an asynchronous server thread. In addition, DEM data tiles can be compressed and kept in compact form. The most important feature of the algorithm in the current context is, however, its ability to easily parallelize terrain rendering on multiple distributed machines.

Our novel parallel terrain rendering solution is based on sort-first and sort-last task decomposition, division of the viewing frustum or DEM database range across several machines respectively and rendering them in parallel. The partial rendering results can then be combined together on a destination display, or multiple displays if desired. We have implemented our parallel terrain renderer using the Equalizer parallel rendering framework [EMP09].

The remainder of the paper is organized as follows. Section 2 briefly discusses the related work in parallel rendering and terrain visualization. In Section 3, we summarize the fundamentals of RASTeR, our basic terrain rendering algorithm. Section 4 discusses the parallelization of RASTeR in detail, and in Section 5, the implementation results are presented. The paper ends with a conclusion in Section 6.

2. Related Work

The early fundamental concepts of parallel rendering have been laid down in [Cro97] and [MCEF94]. Cluster-based parallel rendering has been commercialized for offline rendering for computer generated animated movies or special effects and for other special application domains. Rendering of realistic terrain images on massively parallel computer systems has initially been addressed in [VR91, AG95, LDC96]. However, these approaches are not capable of handling very large data sets at interactive frame rates exploiting current generation GPU hardware. Recent work includes [JLMVK06] which relies on shared resources from a community of users to view 3D data, [YJSZ06] that focuses on rendering on a PC cluster, and [HTMS07] that describes a remote visualization system for large-scale terrain rendering based on a parallel streaming pipeline architecture.

However, none of these previous works on parallel terrain rendering specifically address the problem of rendering task decomposition in the screen (sort-first) or database (sort-last) domain. Moreover, they do not offer a comparative analysis of rendering performance for various rendering modes, configurations and data sizes as presented in this paper.

A number of different multiresolution terrain rendering approaches have been developed in the past, which are reviewed in a recent survey [PG07]. As current graphics hardware can render many million geometric elements per second, focus has shifted from fine-grained LOD rendering to faster block-based LOD selection and terrain rendering techniques [Pom00, Lev02, CGG*03, LPT03, HDJ05, BGP09]. This avoids starvation of the fast GPU by slow detailed LOD selection on the CPU. Despite the increased number of rendered elements, these methods demonstrate the performance advantages of coarse LOD adaption and optimized rendering of batched geometry.

While rendering optimization has focused on optimizing the CPU-GPU communication on a single graphics workstation, none of the methods proposed above address the use of multiple cluster-parallel GPUs for interactive rendering of massive terrain datasets. In this paper we present a distributed cluster-parallel terrain rendering solution based on the RASTeR terrain rendering approach [BGP09] and implemented using the Equalizer parallel rendering framework [EMP09].

3. RASTeR

3.1. Basic Principles

RASTeR [BGP09] is a system for real-time adaptive simplification and rendering of large grid-digital terrain data and follows the idea of batch-based multiresolution triangulation and rendering. It is based on the concept of LOD triangle K-Patches and M-Blocks of elevation data.

A K-Patch is an isosceles right triangle cluster with a constant number of K vertices along each triangle patch boundary. An adaptive batched triangulation can be achieved by selectively splitting K-Patches at their longest edge. These K-Patches can be interpreted as macro triangles of a batched meta bintree and thus can be arranged in a triangle strip sequence. The orientation of a K-Patch is always an instance of one of eight basic isosceles right triangle types. The resolution is doubled for every two consecutive levels in the meta bintree.

An M-Block is a square block of a regular grid of DEM height sample data stored in a file on disk. All M-Blocks are defined to be of equal size, $M \times M$ vertices where $M =$

$2^m + 1$. M-Blocks are organized in a quadtree multiresolution hierarchy. Data within an M-Block can be compressed using standard compression schemes.

M-Blocks and K-Patches form two separate but tightly connected hierarchies, and each meta bintree triangle patch node stores a pointer to the associated quadtree elevation data node as indicated in Figure 1. Exploiting this relationship means that it is possible to clearly separate the LOD selection from rendering, both conceptually as well as in the implementation. The LOD selection is performed on the K-Patch meta bintree, considering a K-Patch as a triangle of triangles. Rendering and resource management are done based on the M-Block quadtree, considering K-Patches as pre-defined triangle strips stored as index buffer over M-Block vertex buffers.

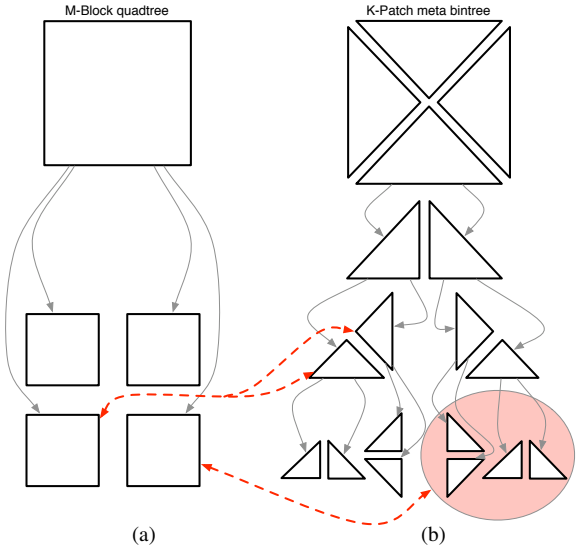


Figure 1: *M-Block height-field quadtree nodes (a) can correspond to different K-Patch meta bintree nodes (b) depending on the selected LOD triangulation. Elevation data in the M-Block quadtree is separated from the triangle mesh connectivity in the K-Patch meta bintree. (Figure from [BGP09].)*

As error metric, a saturated view-dependent definition based on the object-space geometric approximation error is used. This error metric is defined per K-Patch rather than per triangle. For more details on the LOD triangulation and the error metric see [BGP09].

3.2. System

The meta bintree is sufficiently small compared to the DEM height field data and can typically be loaded into main memory. At run-time, the K-Patch bintree is traversed by the LOD manager in the rendering thread, and LOD selection is performed on a per K-Patch basis. To avoid cracks and T-junctions, a saturated octagon error metric [Ger03] is used. The object-space error can thus be quickly computed and compared to a user-specified error tolerance.

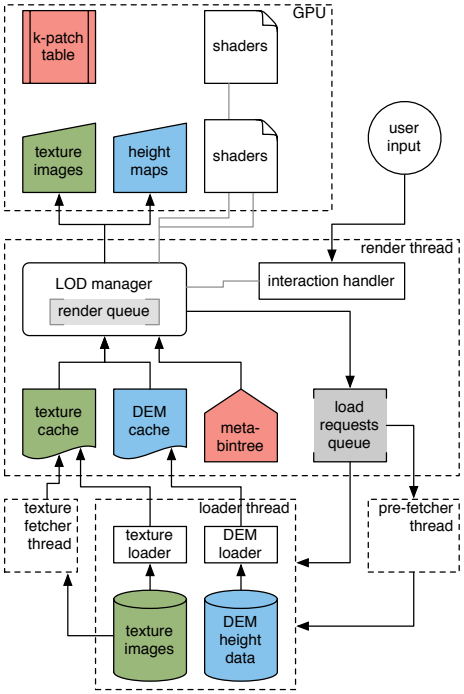


Figure 2: *RASTeR system threads and resource management overview (Figure from [BGP09]).*

To match the K-Patch and M-Block structures, textures are managed in square units and organized in a texture mipmap pyramid. The texture resolution used for one or more related K-Patches is chosen depending on the distance from the camera.

As the K-Patch index buffers are static, the only resources which have to be actively managed during run-time are the M-Block DEM height data and the textures. Loading, caching and prefetching of this data is handled by an asynchronous load queue.

For a more detailed description of RASTeR, please refer to [BGP09].

4. Parallel Terrain Rendering

Our terrain rendering approach is parallelized and implemented using the Equalizer [EMP09] framework for cluster-parallel rendering. Equalizer provides an API and library to facilitate the development of distributed as well as non-distributed parallel real-time rendering applications exploiting multiple GPUs. It is driven by a client-server approach in which the task decomposition and parallel rendering configurations to be executed are independent from the rendering client and entirely managed by the Equalizer server.

In the context of Equalizer, a *node* refers to a single machine (or CPU), a *pipe* to a graphics card, a *window* to an OpenGL drawable and a *channel* to a viewport within a window. For a more detailed description, please see [EMP09]. In the following, we generally assume a canonical arrangement of one *window* per *pipe* and one *pipe* per *node*. For sort-first rendering, there are multiple *channels* per *window*, otherwise one.

At any time, each machine contributing to the rendering executes an independent modified and parallelized RASTeR rendering application. Task distribution is managed by the Equalizer server process according to the user specified configuration. In order to achieve distribution of the rendering task, the modified RASTeR terrain renderer has to take into account either of two parameters supplied by Equalizer: a view frustum or database range for sort-first or sort-last rendering respectively. These parameters are passed to the application nodes by the Equalizer server. Each node uses its individually supplied view frustum for culling, and the database range to select a subset of the entire model to be rendered. All other user parameters, such as pixel-error LOD threshold values and key or mouse controls are duplicated and broadcast to all nodes. However, each parallel instance of the RASTeR application maintains its own rendering front which it incrementally updates every frame.

Since Equalizer has its own OpenGL context handling mechanism and the original implementation of the terrain renderer uses many asynchronous threads, for example to fetch textures and DEM data M-Blocks into GPU memory, we have to provide each application thread the relevant OpenGL context using Equalizer's data and object distribution features.

4.1. Sort-Last or Database Decomposition

In a typical sort-last decomposition, the 3D geometry data is divided among rendering machines, as indicated in Figure 3(a). Final image compositing is based on perspective-correct back-to-front α-blending or z-depth-buffer visibility culling of the partially rendered frames and is performed by Equalizer independently of the client rendering applications.

In order to ensure optimal and scalable parallelization for best performance, it is important that any partitioning scheme ensures that:

1. the rendering primitives or rendering task is divided as equally as possible between the nodes,
2. the per-frame inter communication traffic between the nodes is kept minimal.

The first constraint is not always easy to achieve efficiently as the task of database decomposition is more straightforward in simple rendering as compared to adaptive LOD based rendering. In LOD based visualizations a simple spatial range based decomposition of 3D data fails to distribute the rendering primitives equally among the different

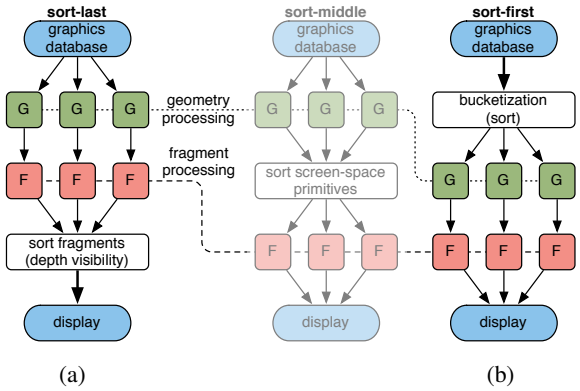


Figure 3: *Sort-last (a), sort-middle and sort-first (b) task decomposition and parallel rendering data flow.*

nodes. Moreover, it may involve a substantial amount of synchronization and communication traffic between the rendering nodes to agree on what part of the constantly changing LOD data is rendered by which node. In the worst case this adversely affects the rendering efficiency from achieving an equal task decomposition.

For sort-last rendering, our terrain renderer decomposes the rendering task by dividing the terrain data into N parts. Equalizer maps the entire 3D database to be displayed to an abstract representation, a linear range interval $[0,1]$. This interval is split into N equal ranges $R_i = [\frac{i}{N}, \frac{i+1}{N}]$ and each rendering node must interpret the range R_i it gets assigned by Equalizer in terms of 3D data to be rendered. All nodes render into a window and channel using the same view frustum. Hence, each node selects and draws $\frac{1}{N}$-th of the (visible) terrain as indicated by R_i, and z-depth visibility compositing is finally done at the destination node by Equalizer.

Using a naive approach, all multiresolution triangles can be enumerated, equally distributed and mapped to one of the N data ranges R_i. Each node must perform LOD selection as well as in-range tests for the selected triangles, and then send the LOD triangles belonging to its own range to the GPU for rendering. Since this involves excessive per-primitive evaluations by the CPU, this would lead to starvation of the GPU. However, as we explain later, RASTeR allows us to perform such evaluations on a coarse level.

As discussed earlier, the bases of RASTeR are triangle K-Patches and terrain data M-Blocks. At rendering time, the LOD manager selects all K-Patches within the given LOD error value range for rendering. These triangle K-Patches in turn activate their corresponding M-Blocks which are the data units containing height and normal values of the terrain. Using these units as a basis for sort-last rendering we discuss three decomposition approaches, each improving upon the other.

4.1.1. Linear Block Enumeration

A simple way to achieve database domain decomposition is to enumerate the terrain M-Blocks and assign them equally to the participating rendering nodes. Thus each node is assigned a linear range R_i of M-Blocks. All nodes traverse the meta bintree in parallel and identify the LOD K-Patches to be displayed. Each node checks if the selected K-Patches correspond to an M-Block within its own range R_i, and only this filtered set of K-Patches is then rendered. For example, if the range of the current node as supplied by Equalizer is $R_i = [l,r]$, with the origin of an M-Block is given by $O_{\mathrm{M}}(x,y)$ and x_{max} being the maximum x-dimension value, then $l * x_{max} \leq O_{\mathrm{M}}(x) \leq r * x_{max}$ can be checked to decide if a K-Patch should be rendered on this node, see also Figure 4(a). In fact, as soon as a K-Patch fails this test, the traversal of the meta bintree can be stopped, as the child nodes do not fall in the given range either. With this approach, we can exploit the K-Patches as triangle cluster units instead of doing per primitive LOD and in-range queries.

The drawbacks with this approach are that the meta bintree LOD traversal has to be constrained to the branches corresponding to the M-Blocks indicated by R_i, and that this LOD traversal and selection are easily susceptible to view changes upon rotation or translation. Above all, this approach cannot guarantee an even division of data across all machines, since the adaptive view-dependent LOD selection unequally maps to the fixed assignment of M-Blocks per node, and hence is not the best for parallel rendering (see also Figure 6(a)).

4.1.2. Quadtree Enumeration

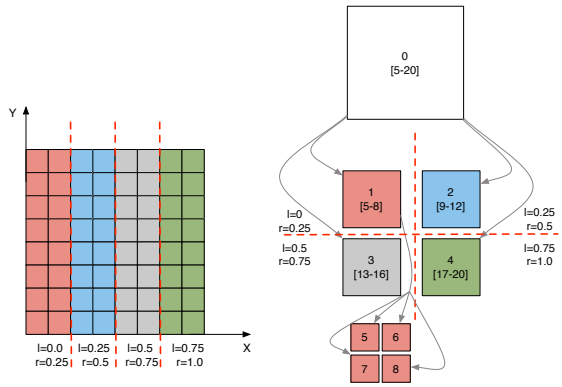
An improved approach makes use of the quadtree structure of the M-Block hierarchy (see Figure 4(b)). Starting from the root M-Block, all quadtree nodes are recursively enumerated. Bottom-up, intervals to all internal nodes are assigned that cover the range of its descendants. At runtime, all bintree K-Patches that correspond to M-Blocks in the range $R_i = [l,r]$ supplied by Equalizer are selected for rendering on a particular machine. The range test is simple and can be made as follows:

$$\begin{aligned} l * n_{max} &\leq L_{\mathrm{M}} \leq r * n_{max} \vee \\ l * n_{max} &\leq R_{\mathrm{M}} \leq r * n_{max} \end{aligned}$$

where n_{max} refers to the maximum number of a leaf node and $[L,R]_{\mathrm{M}}$ is the interval of the M-Block node itself covering its descendants. This decomposition strategy allows us to select coherent terrain data per machine that is not as susceptible to rotational and translational changes as in the previous approach. Figure 6(b) clearly demonstrates the advantage of this improved decomposition mode over a simple linear enumeration.

4.1.3. Active K-Patch Enumeration

The problem with both of the above decomposition schemes is that they do not ensure that all rendering machines get a similar amount of rendering workload and hence optimal performance is not reached. Since our basic rendering units are triangle K-Patches and M-Blocks, any simple spatial division of these units cannot guarantee that the rendering load is evenly distributed across all machines in terms of the number of drawing primitives. This can however, be achieved by making the observation that each K-Patch contains the same number of triangles. Therefore, an optimal task decomposition can be achieved by dividing the list of visible K-Patches equally among all rendering nodes as illustrated in Figure 5. After the meta bintree LOD traversal, the front of visible K-Patches is the same across all machines. Each machine can choose from this list a particular sub-set of K-Patches to render. Thus, the front of selected K-Patches is enumerated and mapped to the ranges R_i provided by Equalizer. Using this view-adaptive assignment of visible triangle K-Patches, each node can select a similar number of geometric primitives to display without the need for any communication overhead. A run-time view is shown in Figure 6(c).



(a) (b)

Figure 4: *Sort-last database decomposition of terrain for four nodes using (a) linear block and (b) quadtree enumeration.*

4.2. Sort-First or Screen Decomposition

The sort-first decomposition mode involves task division in screen space and is relatively simple (see also Figure 6(d)). For each frame, before the LOD meta bintree is traversed, every rendering node updates its view frustum parameters to the ones indicated by the Equalizer server. The meta bintree traversal is then restricted to the particular view frustum, performing view-frustum culling of the LOD meta bintree on that node. Since in sort-first mode different machines render different parts of terrain that occupy mutually separate parts of the screen, final image assembly is simple and fast as it does not involve any costly z-depth or α-compositing stage.

5. Results

Equalizer and RASTeR are both written in C++ using GLSL shaders. The implementation is tested on a 10-node Linux

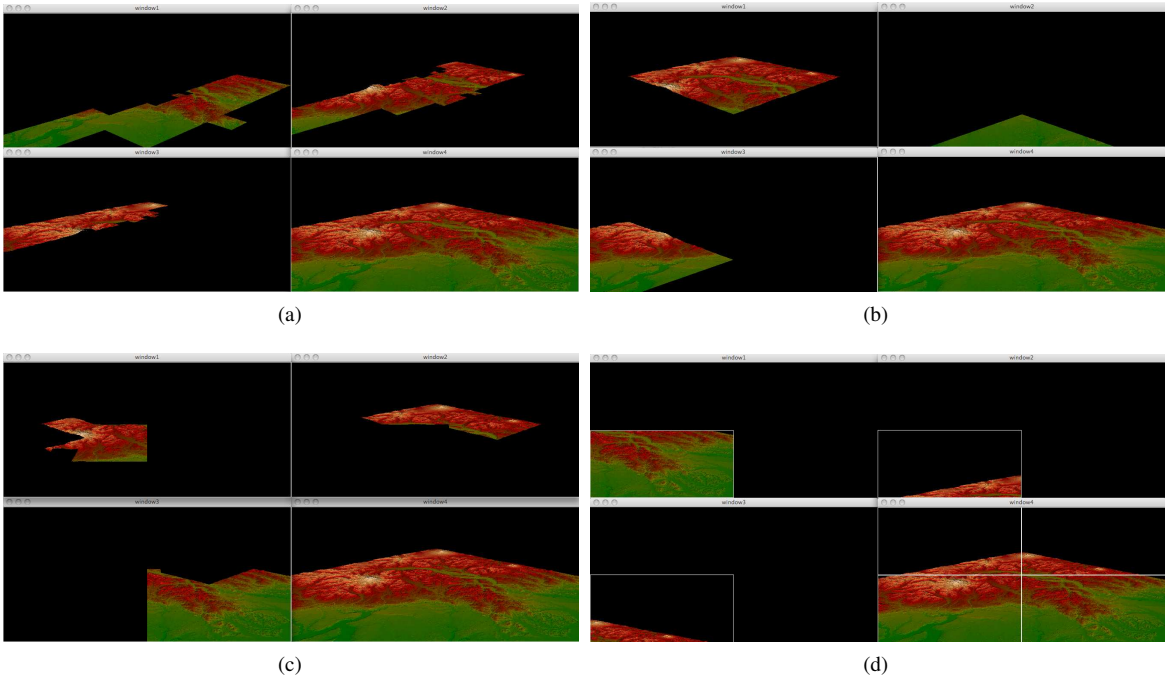


Figure 6: *Screenshots of sort-last database decomposition of terrain on four nodes using (a) linear block, (b) quadtree and (c) active K-Patch enumeration. (d) Sort-first view frustum decomposition.*

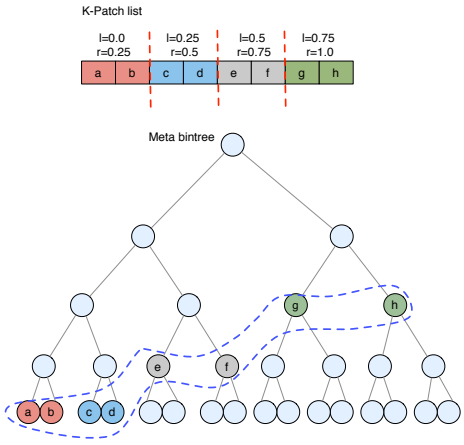
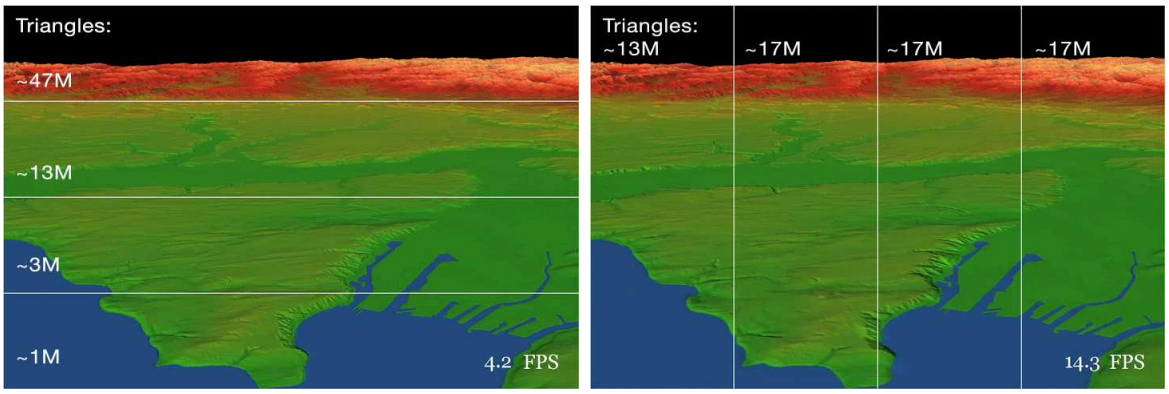


Figure 5: *Sort-last database decomposition for four nodes using active K-Patch enumeration.*

cluster with 2 Gbit/s Myrinet for image compositing and 1 Gbit/s network for out-of-core terrain data retrieval. Each node features a dual 2.2 GHz AMD Opteron CPU, 4GB of RAM and GeForce 9800 GX2 graphics. Two different data sets were used: Puget Sound (16k×16k vertices) and SRTM (32k×32k vertices); all tests were carried out using 1280×1024 pixels viewport.

The linear block and quadtree enumeration sort-last data decomposition modes should be used in combination with dynamic load balancing to distribute the rendered geometry as evenly as possible. Otherwise, a static assignment of data to rendering nodes cannot achieve scalable sort-last rendering at all. Current sort-last load balancing in Equalizer redistributes the data ranges R_i based on measures of time spent on rendering by all nodes in the previous frame. This type of load balancing based on past performance can easily and quickly be integrated into a sort-last rendering system. However, this scheme does not take into account caching of data in main memory and can thus tend to shift load towards a single machine, which has most of the data cached already. Awarding a fast rendering node with further data to be displayed can lead to most of the geometry being rendered on one node, while other machines are busy updating their caches, resulting in poor overall performance.

Our tests have shown that linear block and quadtree enumeration with the above outlined fixed or past-frame adaptive load distributions do not provide scalable sort-last rendering. Only our third approach using active front K-Patch enumeration showed performance improvements when adding more rendering nodes. This method provides simple and automatic load balancing (since all nodes render almost equal number of triangles) which is not based on past rendering times, while other approaches require more sophisticated load balancing computations. We demonstrate



(a) (b)

Figure 7: *Sort-first screen partitioning causes a less even distribution of data to be rendered in the case of vertical (a) compared to horizontal (b) partitioning. (four nodes)*

Figure 8: *Zurich dataset using display wall configuration*

sort-last scalability using this technique in the graphs below and in the accompanying video.

For sort-first parallel rendering we have analyzed two simple screen decomposition modes, vertical and horizontal tiling, that partition the view frustum equally. Vertical partitioning leads to wide screen regions as shown in Figure 7(a) and can lead to dramatically uneven data distribution per tile due to the perspective projection. Despite view-dependent LOD, the top tiles can receive significantly more geometry per screen space area and thus will limit the overall frame rate. Horizontal partitioning as in Figure 7(b) leads to a much more even distribution of geometry per screen tile. For larger numbers of rendering nodes, however, both vertical and horizontal tiling lead to large aspect ratio screen tiles with poor parallel rendering scalability. Hence, a more regular rectangular screen partitioning is necessary for sort-last rendering on many nodes (see also Figure 6(d)).

Figures 9(a) and 9(c) present frame rate graphs for moving forward and turning camera trajectories, while Figures 9(b) and 9(d) present those for the camera zooming into the terrain. As we can see from these graphs, in both sort-first (2D) and sort-last (DB) parallel rendering modes pure drawing performance (labeled as 2D - Rendering and DB - Rendering respectively) scales at least linearly. Superlinear performance of rendering can be explained by reduced data fetching since each machine fetches only those terrain M-Blocks that are not already cached locally in main memory. The reduced size of the rendering front of active K-Patches on a single machine allows it to be cached more efficiently in GPU and main memory, hence avoiding repeated data fetch-

ing. Pure sort-last rendering scales better than sort-first because each machine renders a similar number of triangles, thus data is well distributed among them. However, this cannot be ensured in the case of sort-first task decomposition.

Overall rendering performance depends largely on the compositing stage of the parallel rendering framework, which includes reading of partial images back from GPUs, transmitting them to the destination node and assembling final frames for display. The decrease of the final performance (labeled as 2D and DB) with increasing number of nodes on Figure 9 happens due to the image throughput bottleneck. The amount of data that has to be sent over the network in case of sort-last rendering and compositing is roughly twice larger than for sort-first, thus network saturation happens earlier despite the rendering itself being faster. In our case, sort-last network saturation happens at around 15 fps, which is independent of the drawing speed. That means if the initial rendering on one node is already fast, overall performance will not scale well with more rendering nodes. For the smaller Puget Sound terrain model, when the camera is zooming in and the initial speed is about 3 fps, sort-last rendering does not scale anymore after 6 nodes (see also Figure 9(b)). For sort-first rendering, network saturation should happen at around 30 fps. This speed is not reached in our experiments, therefore, the final performance of sort-first rendering scales almost linearly up to the tested number of nodes.

Figure 9 demonstrates that performance of distributed RASTeR rendering scales very well. Overall performance however is mostly limited by network throughput and by the compression used for partial images. Therefore, it is possible to improve overall performance if better network or more sophisticated compression schemes are used, especially in case of sort-last rendering.

The proposed K-Patch enumeration sort-last parallel rendering solution works well and does not interfere with Equalizer's general task distribution and parallel rendering approach. The task decomposition flexibility of Equalizer is fully maintained, an example showing a tiled wall configuration driven by our parallel terrain rendering application is given in Figure 8.

6. Conclusion

In this paper a new perspective on real-time multiresolution out-of-core terrain visualization in the context of scalable cluster-parallel rendering has been presented. We have shown that scalable parallel rendering cannot easily be achieved by simple sort-last or sort-first task distribution, even if applied to adaptive LOD terrain rendering approaches. To achieve effective parallel rendering using distributed graphics hardware resources over a network, more aspects on task assignment and out-of-core loading have to be taken into account. This is the case because the bottleneck shifts from the simple CPU-GPU communication problem to a more complex and challenging environment of distributed resources.

The experiments on sort-last and sort-first parallel terrain rendering demonstrate possible pitfalls if adapting known out-of-core LOD, but single CPU-GPU terrain visualization methods to a parallel system. In particular, we have introduced a novel sort-last data decomposition technique that achieves per-frame automatic load balancing. While this method realizes highly scalable, multiresolution terrain rendering from out-of-core for very large grid-digital elevation models, the introduced technique and analysis only presents the first steps towards efficient cluster-parallel terrain rendering. More challenges have to be addressed, in particular towards the analysis and reduction of system-wide overhead when using a very large number of parallel rendering nodes.

Acknowledgements

This work was supported in part by the Swiss National Science Foundation under Grant 200021-116329/1. The authors would like to thank and acknowledge the following institutions and projects for providing 3D data sets: the Georgia Tech Large Geometric Models Archive for the Puget Sound data set, NGA and NASA for providing the SRTM height field. as well as swisstopo for the DHM25 model of Zürich (used in the video). We would also like to thank Stefan Eilemann for his help on using Equalizer in this project.

References

[AG95] AGRANOV G., GOTSMAN C.: Algorithms for rendering realistic terrain image sequences and their parallel implementation. *The Visual Computer 11*, 9 (1995), 455–464.

[BGP09] BÖSCH J., GOSWAMI P., PAJAROLA R.: RASTeR : Simple and efficient terrain rendering on the GPU. In *Proceedings EUROGRAPHICS Areas Papers* (2009), pp. 35–42.

[CGG*03] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: BDAM - batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (2003), 505–514.

[Cro97] CROCKETT T. W.: An introduction to parallel rendering. *Parallel Computing 23* (1997), 819–843.

[EMP09] EILEMANN S., MAKHINYA M., PAJAROLA R.: Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics 15*, 3 (May/June 2009), 436–452.

[Ger03] GERSTNER T.: *Top-Down View-Dependent Terrain Triangulation using the Octagon Metric*. Tech. rep., Institute of Applied Mathematics, University of Bonn, 2003.

[HDJ05] HWA L. M., DUCHAINEAU M. A., JOY K. I.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 355–368.

[HTMS07] HU C., TIAN J., MING D., SHEN D.: Multi-screen tiled displayed, parallel rendering system for a large terrain dataset. In *MIPPR 2007,Medical Imaging, Parallel Processing of Images, and Optimization Techniques, Vol:6789* (2007), pp. 47–54.

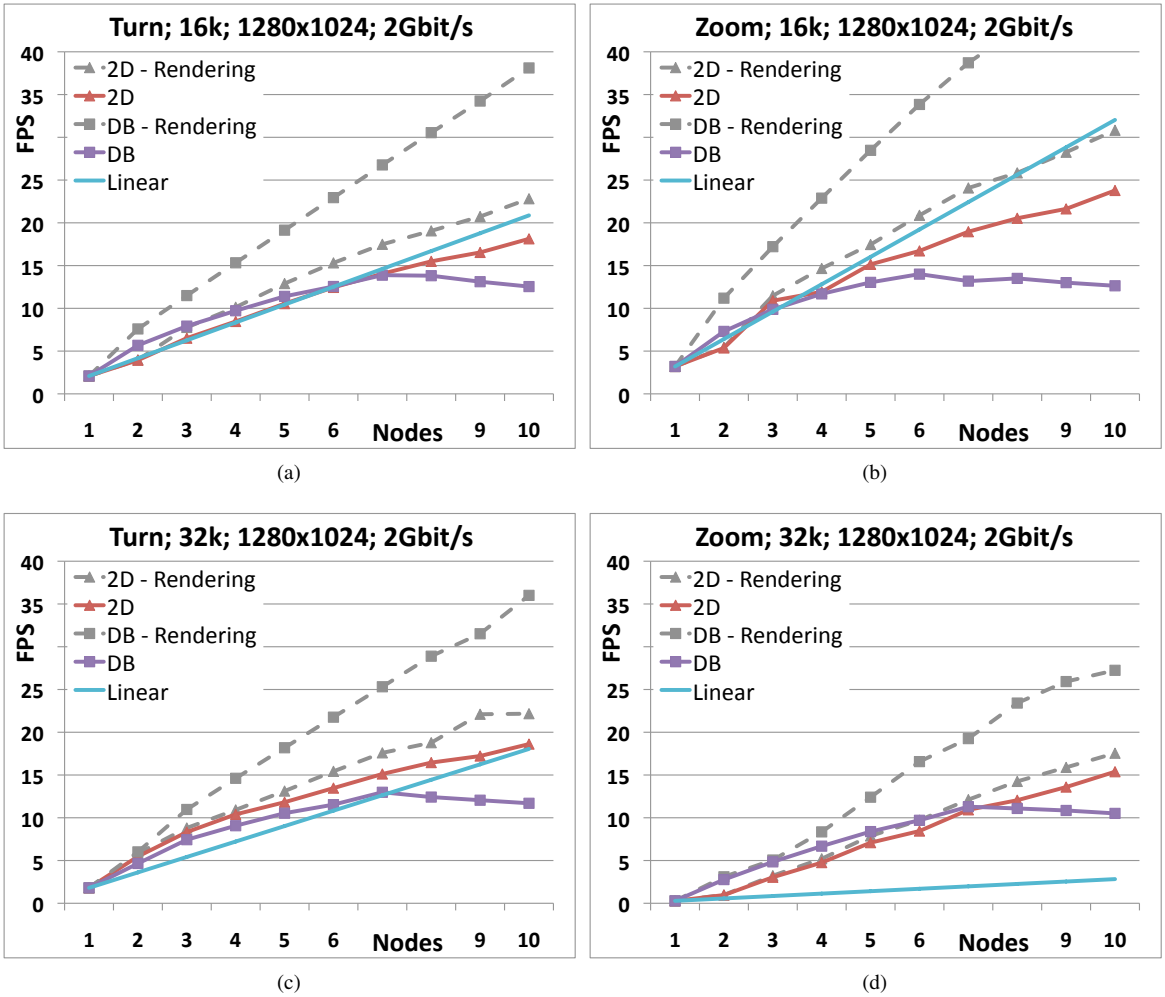


Figure 9: *Graphs showing rendering performance on 10 machines in parallel using DEM models of (a),(b) 16k×16k Puget Sound and (c),(d) 32k×32k SRTM grids with camera in turning and zooming trajectories respectively. 2D - Rendering refers to sort-first rendering, 2D to sort-first rendering with compositing, DB - Rendering to sort-last rendering, DB to sort-last rendering with compositing.*

[JLMVK06] JOHNSON A., LEIGH J., MORIN P., VAN KEKEN P.: GeoWall: Stereoscopic visualization for geoscience research and education. *IEEE Computer Graphics and Applications 26*, 6 (November-December 2006), 10–14.

[LDC96] LI P. P., DUQUETTE W. H., CURKENDALL D. W.: RIVA: A versatile parallel rendering system for interactive scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 2*, 3 (1996), 186–201.

[Lev02] LEVENBERG J.: Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization* (2002), Computer Society Press, pp. 259–266.

[LPT03] LARIO R., PAJAROLA R., TIRADO F.: Hyperblock-QuadTIN: Hyper-block quadtree based triangulated irregular networks. In *Proceedings IASTED International Conference on Visualization, Imaging and Image Processing (VIIP)* (2003), pp. 733–738.

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (1994), 23–32.

[PG07] PAJAROLA R., GOBBETTI E.: Survey on semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer 23*, 8 (2007), 583–605.

[Pom00] POMERANZ A. A.: *ROAM Using Surface Triangle Clusters (RUSTiC)*. Master's thesis, University of California at Davis, 2000.

[VR91] VEZINA G., ROBERTSON P. K.: Terrain perspectives on a massively parallel SIMD computer. In *Proceedings Computer Graphics International (CGI)* (1991), pp. 163–188.

[YJSZ06] YIN P., JIANG X., SHI J., ZHOU R.: Multi-screen tiled displayed, parallel rendering system for a large terrain dataset. In *International Journal of Virtual Reality, 5(4)* (2006), pp. 47–54.